CS839: AI for Scientific Computing
Scientific Computing Basics

Misha Khodak

University of Wisconsin-Madison

29 January 2026

Announcements

Enrollment:

- Finalized this week. Please keep checking your status.

Outline

- **Partial differential equations**
  - examples, classification, initial / boundary conditions
- **Numerical simulations**
  - solvers, meshing, stability, linear systems
- **Applications**
  - forward problems, inverse problems, control problems

Outline

- **Partial differential equations**
  - examples, classification, initial / boundary conditions
- **Numerical simulations**
  - solvers, meshing, stability, linear systems
- **Applications**
  - forward problems, inverse problems

What is a partial differential equation (PDE)?

An equation involving partial derivatives of an unknown function $u(x_1, x_2, \ldots, t)$

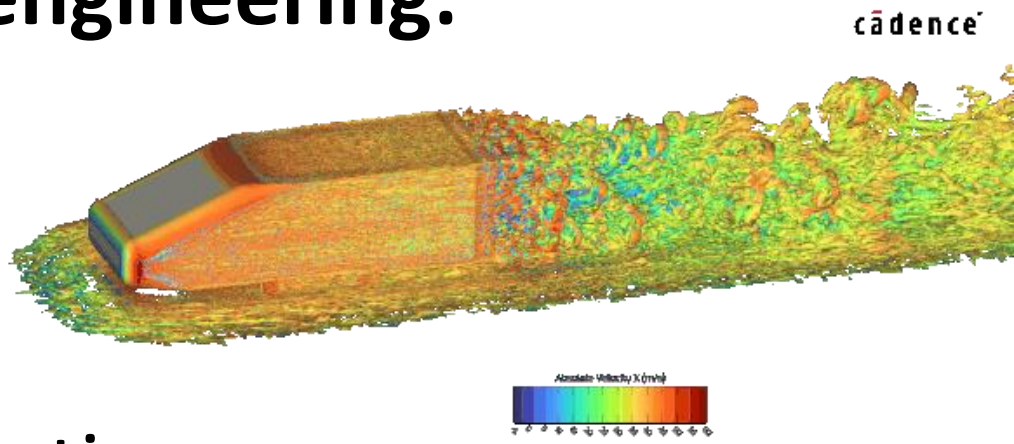
Examples:

name	equation	type
heat equation	$\partial_t u = \alpha \nabla^2 u$	parabolic
wave equation	$\partial_t^2 u = c^2 \nabla^2 u$	hyperbolic
Laplace equation	$\nabla^2 u = 0$	elliptic
Poisson equation	$\nabla^2 u = f$	elliptic

Here $\nabla^2 = \partial_x^2 + \partial_y^2 + \cdots$ is called the **Laplacian**

Why PDEs matter

They are everywhere in science & engineering:



1. fluid dynamics: Navier-Stokes
2. heat transfer: diffusion equation
3. electromagnetics: Maxwell's equations
4. quantum mechanics: Schrodinger equation
5. finance: Black-Scholes equation
6. biology: reaction-diffusion equation

Classification of (2nd-order) linear PDEs

The general form:

$$A\partial_x^2 u + 2B\partial_{xy} u + C\partial_y^2 u + \text{lower order terms} = 0$$

Classification via the discriminant $\Delta = B^2 - AC$:

discriminant	type	prototype	physical behavior
$\Delta < 0$	elliptic	Laplace	equilibrium, steady-state
$\Delta = 0$	parabolic	heat equation	diffusion / smoothing
$\Delta > 0$	hyperbolic	wave equation	propagation, finite speed

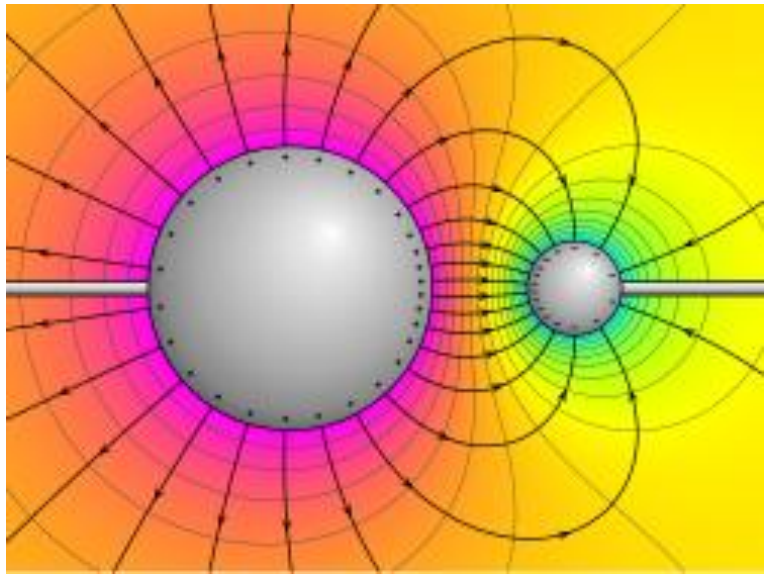
Elliptic PDEs

steady-state / equilibrium problems

$$\nabla^2 u = f \text{ (Poisson equation)}$$

Characteristics:
- information propagates in all directions
- solutions are typically smooth

Applications:
- electrostatics
- gravitational potential
- heat diffusion
- incompressible flow (pressure)



Parabolic PDEs

diffusion problems

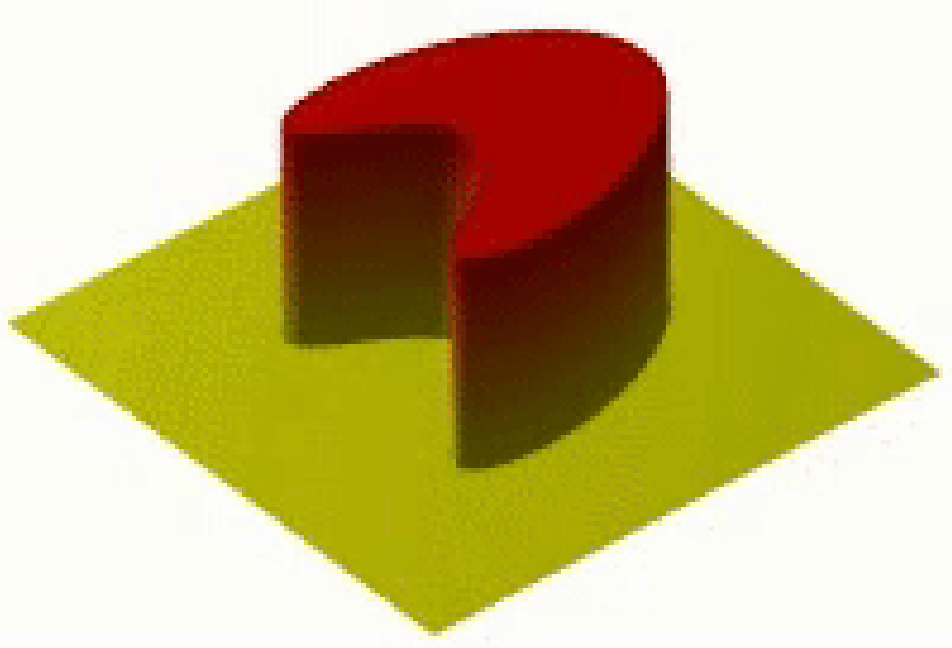
$$\partial_t u = \alpha \nabla^2 u \text{ (heat equation)}$$

Characteristics:

- time-dependent
- solutions smooth out over time

Applications:

- heat conduction
- mass diffusion
- option pricing
- image denoising



Hyperbolic PDEs

wave propagation problems

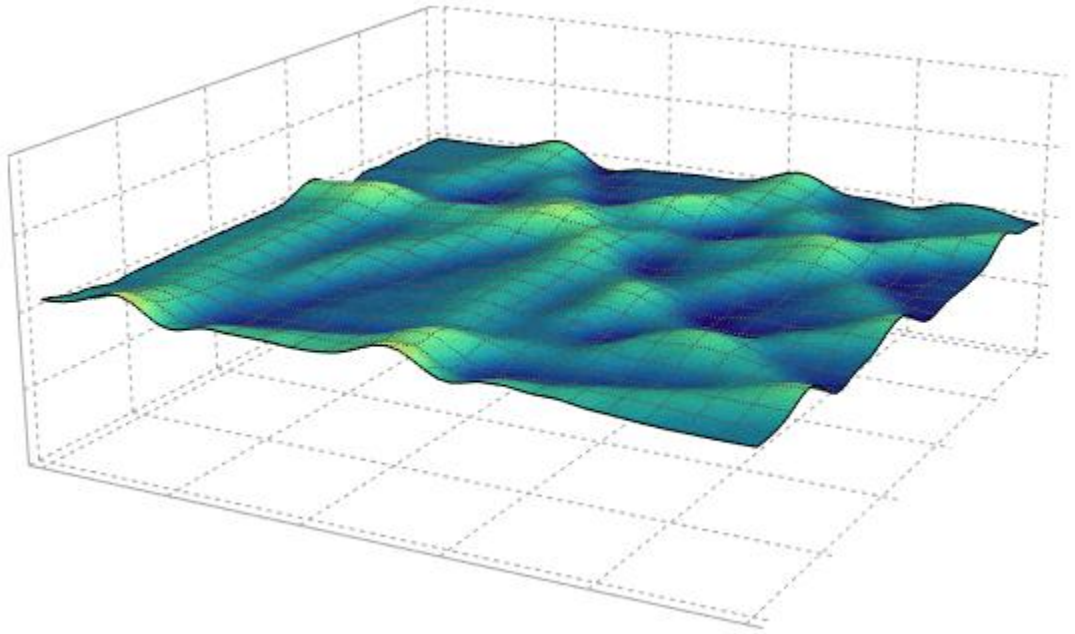
$$\partial_t^2 u = c^2 \nabla^2 u \text{ (wave equation)}$$

Characteristics:

- characteristic curves along which information propagates
- develops discontinuities (shocks)

Applications:

- acoustics
- electromagnetics
- seismic waves
- compressible fluid dynamics



Specifying the problem: The domain

The spatial region where we seek the solution is the domain Ω

type	description	example
bounded	finite region with a boundary	flow in a pipe, heat in a plate
unbounded	extends to infinity	wave radiation, free-space scattering
periodic	wraps around (no boundary)	toy problems, climate models

Geometry complexity:

- simple, e.g. rectangles, sphere: analytical tools available
- complex, e.g. airfoils, porous media: need numerical algos

The domain determines what boundary conditions are needed, how to generate the mesh, and compute cost.

Specifying the problem: Boundary conditions

What happens at the edge $\partial\Omega$ of Ω

type	mathematical form	physical meaning
Dirichlet	$u = g$ on $\partial\Omega$	prescribed value
Neumann	$\partial_n u = h$ on $\partial\Omega$	prescribed flux
Robin	$\alpha u + \beta \partial_n u = \gamma$ on $\partial\Omega$	convective heat transfer

Other important conditions:

- periodic
- outflow
- Interface between materials

Specifying the problem: Initial conditions

To solve a transient PDE we need to specify what happens at $t = 0$ via some (temporally constant) function $u_0(x)$

$$u(x, 0) = u_0(x)$$

Often we try to set a uniform / simple initialization, but this can be difficult in irregular domains.

Outline

- **Partial differential equations**
  - examples, classification, initial / boundary conditions
- **Numerical simulations**
  - solvers, meshing, stability, linear systems
- **Applications**
  - forward problems, inverse problems

How do we solve PDEs numerically?

typically we try to convert a continuous PDE to a system of algebraic equations

method	idea	use-case
finite difference method	Replace derivatives with difference quotients	simple geometries, regular grids
finite element method (FEM)	weak formulation + basis functions	complex geometries, unstructured meshes
finite volume method	integral conservation on control volumes	conservation laws
spectral	global basis functions	smooth solutions, periodic BCs

Finite difference method

Derive from a Taylor series expansion:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(x) + O(h^3)$$

derivative	approximation	accuracy order
forward	$f'(x) = \frac{f(x+h) - f(x)}{h}$	$O(h)$
backward	$f'(x) = \frac{f(x) - f(x-h)}{h}$	$O(h)$
central	$f'(x) = \frac{f(x+h) - f(x-h)}{2h}$	$O(h^2)$
second-derivative	$f'(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$	$O(h^2)$

Finite difference method: Heat equation

1D heat equation: $\partial_t u = \alpha \partial_{xx} u$

Discretization with resolution Δx in space and Δt in time:

$$\frac{u(x_j, t_{n+1}) - u(x_j, t_n)}{\Delta t} = \frac{u(x_{j+1}, t_n) - 2u(x_j, t_n) + u(x_{j-1}, t_n)}{(\Delta x)^2}$$

So we can set:

$$u(x_j, t_{n+1}) = u(x_j, t_n) + r\left(u(x_{j+1}, t_n) - 2u(x_j, t_n) + u(x_{j-1}, t_n)\right)$$

Here $r = \alpha \Delta t / (\Delta x)^2$

Finite element method (FEM)

Steps:

1. mesh the domain
   - discretize the domain Ω into triangle / tetrahedra
   - place nodes at vertices

2. approximate the solution
   - assume $u = \sum_i U_i \phi_i(x)$
   - U_i are unknown values at nodes
   - ϕ_i are simple functions

3. convert to a linear system and solve for the nodal values

Finite volume method

Often we care strongly about **conservation** (of mass, momentum, energy). Finite difference and FEM does not guarantee this, but **finite volume** methods do:

1. divide domain into cells V

2. store average value of u in each cell (not point values)

3. track fluxes across cell faces:
   - flux on the cell of one face must correspond to a reverse flux on the face of another cell
   - forces a change in conserved quantity in one cell must correspond to a reverse change in that quantity in another

Spectral methods

Approximate solution via a global basis functions:

$$u_N(x) = \sum_{k=0}^{N} \hat{u}_k \phi_k(x)$$

Common choices:

- Periodic domains: Fourier (e^{ikx})
- Non-periodic smooth solutions: Chebyshev polynomials
- Other: Legendre polynomials

Advantages: exponential convergence / can be very fast (FFTs)
Disadvantages: strong constraints (smoothness, periodic BCs)

When do we use which method?

simulation property	finite differences	finite element	finite volume	spectral
geometry	simple	complex	moderate	simple
conservation	none	weak	exact	none
accuracy	low	flexible	moderate	high
smoothness	maybe	ok	good	poor

Timestepping

The previous schemes focus on *spatial* discretization. We typically add time discretization by converting the problem to a **system of ODEs** via the **method of lines**:

$$\partial_t u = L[u] \rightarrow \partial_t \mathbf{u} = \mathbf{f}(\mathbf{u}, t)$$

where

- $\mathbf{u}(t) = [u_1(t), u_2(t), \ldots, u_N(t)]$ are nodal values
- $\mathbf{f}$ encodes the spatial operator (e.g. $\mathbf{fu} = \mathbf{Au}$ for matrix $\mathbf{A}$)

Now we can consider many schemes for ODE solving…

How do we march forward in time?

$\partial_t \mathbf{u} = \mathbf{f}(\mathbf{u}, t)$ with $\mathbf{u}(0) = \mathbf{u}_0$

explicit methods	implicit methods
$\mathbf{u}(t+1) = \mathbf{u}(t) + \Delta t \mathbf{f}(\mathbf{u}(t), t)$	$\mathbf{u}(t+1) = \mathbf{u}(t) + \Delta t \mathbf{f}(\mathbf{u}(t+1), t)$
update on LHS only	update on both sides
no system solve	must solve a (non)linear system
conditionally stable	unconditionally stable
small Δt required	can take large timesteps

Explicit schemes

Advantages:

- Simple to implement
- Cheap per timestep
- Easy to parallelize

Disadvantages:

- have to take many timesteps to maintain stability
- e.g. by following the CFL condition: $\frac{c\Delta t}{\Delta x} \leq C$
  - c is the wave / signal speed
  - C is method-dependent but often 1

Good for cases where you have to take a small timestep for other reasons or you want to run on GPU.

Examples: forward Euler, RK4

Implicit schemes

Advantages:

- Unconditionally stable
- Can take large time steps

Disadvantages:

- have to solve a (non)linear system at each timestep
- harder to parallelize

Good for stiff problems (multiple time scales)

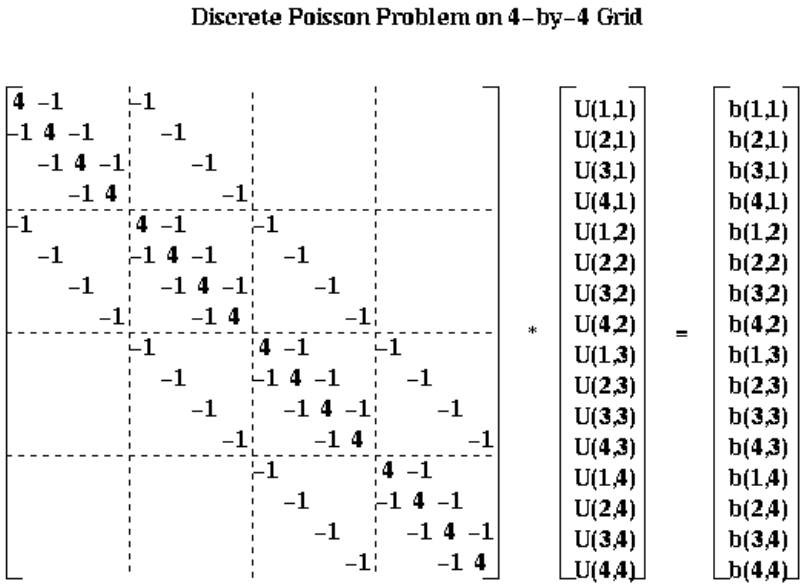
Examples: backward Euler, Crank-Nicholson

Can even combine implicit + explicit (IMEX schemes)

Linear system solvers

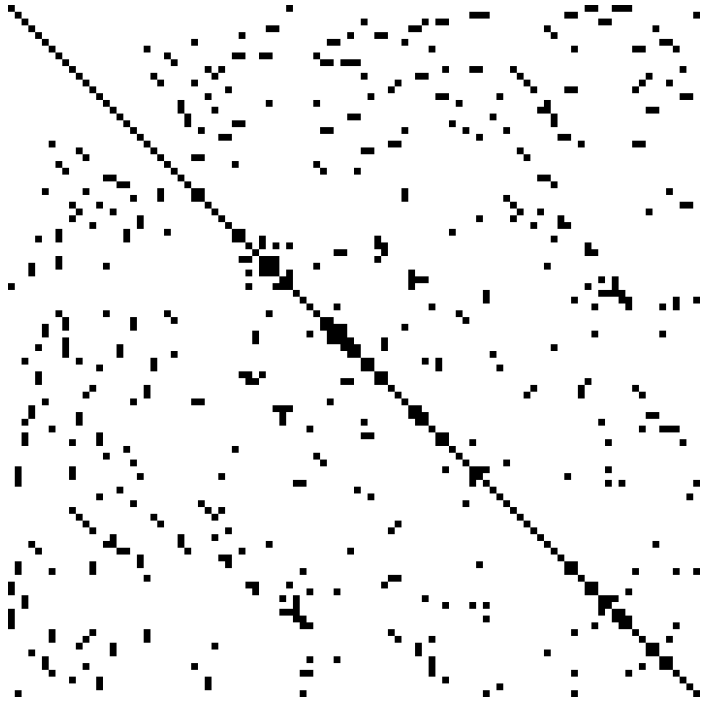
Discretizing often converts task to solving $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A}$ is

- large (millions to billions)
- sparse (mostly zero, $\mathrm{nnz}(\mathbf{A}) = O(n)$)
- structured (banded, block-structured, etc.)



Berkeley EECS

FEM on a disk



Wikipedia

Can we solve $\mathbf{Ax} = \mathbf{b}$ exactly?

Suppose $\mathbf{A}$ is symmetric positive-definite (SPD)

Can try Cholesky decomposition: compute matrix $\mathbf{L}$ such that

- $\mathbf{L}\mathbf{L}^\top = \mathbf{A}$
- $\mathbf{L}$ is ***triangular***

Then can solve $\mathbf{L}\mathbf{L}^\top\mathbf{x} = \mathbf{b}$ using triangular solves in $O(\mathrm{nnz})$ time

However: $\mathbf{L}$ is not necessarily sparse if $\mathbf{A}$ is sparse, and computing it takes worst case $O(n^3)$ time

Still useful if:

- we have a sequence of medium-sized system with same LHS $\mathbf{A}$
- we need to use it as a subroutine, e.g. of a preconditioner

Iterative solvers

- use matrix-vector products to construct a sequence of iterates $\mathbf{x}_1, \ldots,$ that converges to the solution
- stop when $\|\mathbf{A}\mathbf{x}_k - \mathbf{b}\| \leq \varepsilon$ (typically very small)
- most methods are Krylov subspace methods that output a solution in the Krylov subspace $\mathbf{b}, \mathbf{A}\mathbf{b}, \mathbf{A}^2\mathbf{b}, \ldots$
  - conjugate gradient (for SPD matrices)
  - GMRES (for general matrices)
  - …
- performance typically depends on spectral properties of the matrix (condition number, clustering of eigenvalues, etc.)

Preconditioning

what if $\mathbf{A}$ is ill-conditioned, i.e. $\kappa(\mathbf{A}) = \|\mathbf{A}\|\|\mathbf{A}^{-1}\|$ is large?

Krylov methods almost always use a preconditioner $\mathbf{M}$:

- $\mathbf{M} \approx \mathbf{A}$
- $\mathbf{M}^{-1}$ is fast to compute (e.g. $O(\mathrm{nnz})$ time)
- converts problem to solving $\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}$
- often defined implicitly

Common choices:

- Jacobi ($M = \mathrm{diag}(\mathrm{diag}(\mathbf{A}))$) and its blocked variants
- incomplete Cholesky (IC(0)) and its thresholded versions
- decomposition approaches
- **multigrid**

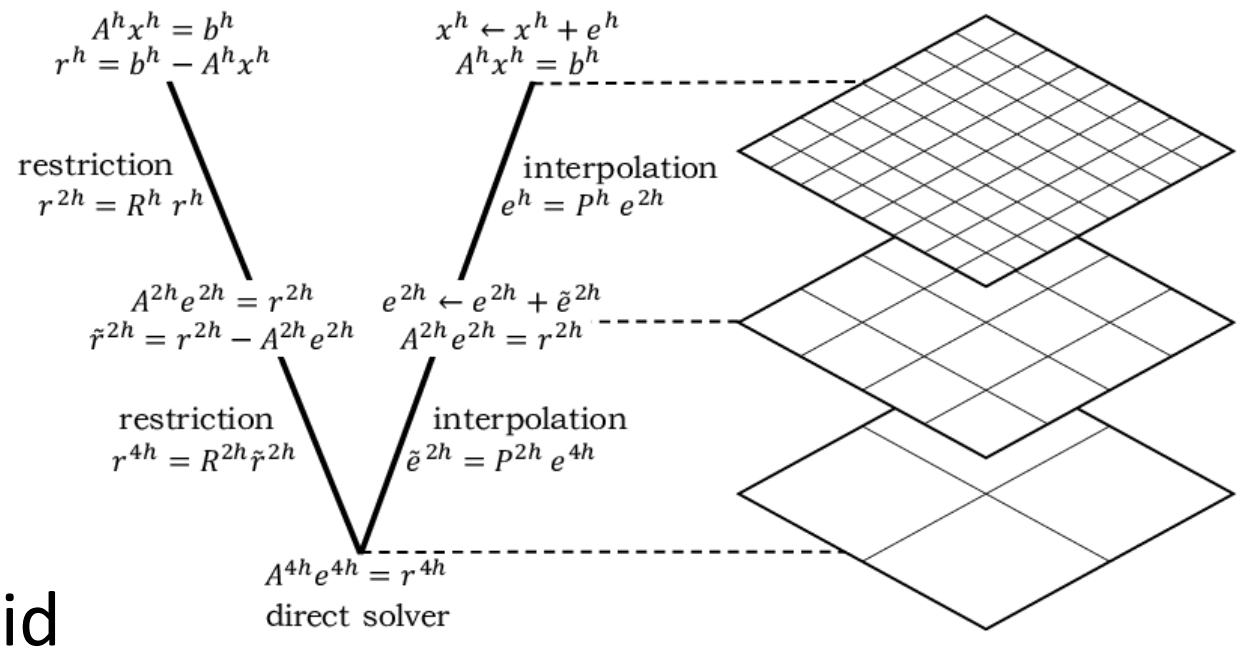
Multigrid

Solvers are good at smoothing high-frequency errors

Idea:

- **smooth** on fine grid
- **restrict** residual to coarser grid
- **solve** recursively on coarse grid
- **interpolate** back to fine grid
- **smooth** again

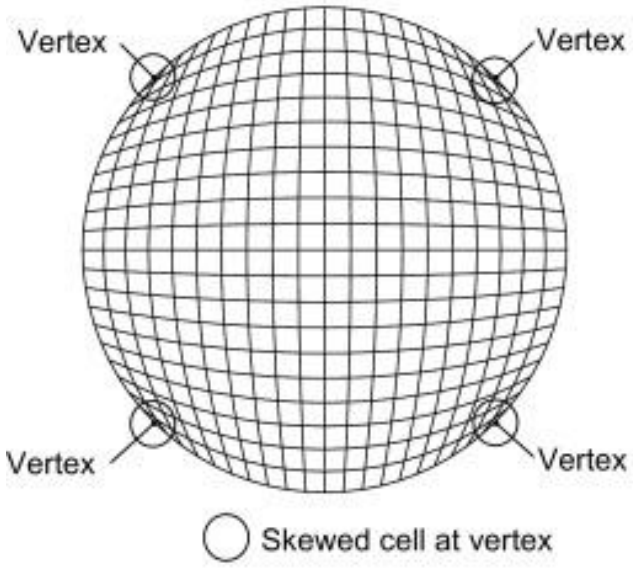
Optimal $O(n)$ complexity for certain classes of problems



Meshing

The mesh size/quality significantly affects simulation speed:

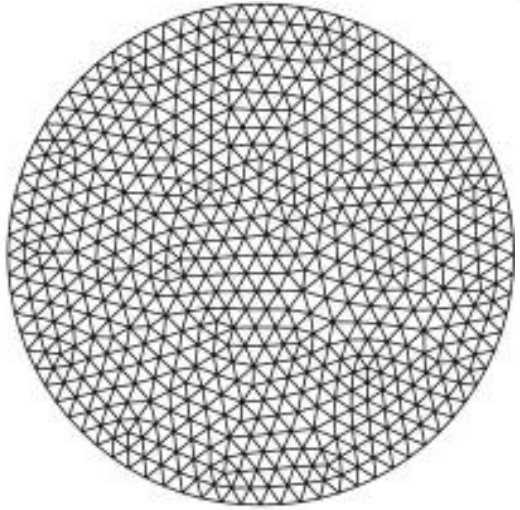
- determines accuracy / compute cost
- must resolve important features (boundary layers, shocks)
- affects matrix conditioning



Structured mesh

Two main types:

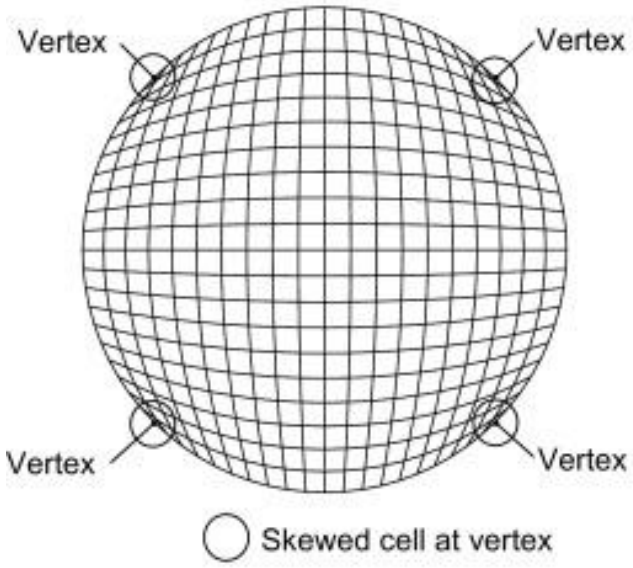
- Structured: regular grid, easy to use/index, limited use-cases
- Unstructured: handles any geometry, can have better quality metrics



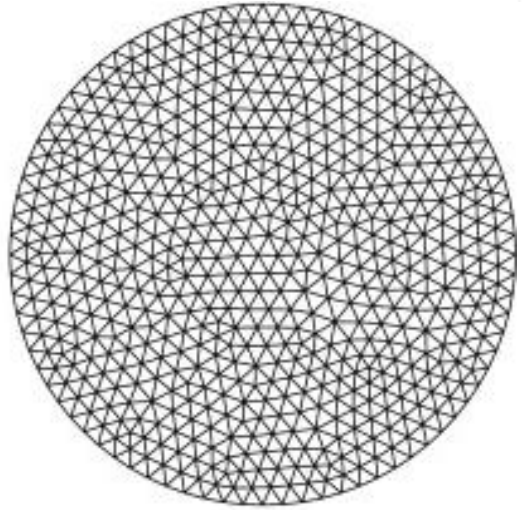
Unstructured mesh

What makes a good mesh?

- low aspect ratio (ratio of longest to shortest edge)
- smooth transitions between element sizes
- captures the physics required (typically needs to be finer in areas of higher activity)



Structured mesh



Unstructured mesh

Outline

- **Partial differential equations**
  - examples, classification, initial / boundary conditions
- **Numerical simulations**
  - solvers, meshing, stability, linear systems
- **Applications**
  - forward problems, inverse problems

Forward problem

Given PDE and initial/boundary conditions, find a solution $u(x, t)$

Applications:

- prediction (e.g. weather, system behavior)
- uncertainty quantification (propagate IC / parameter errors)

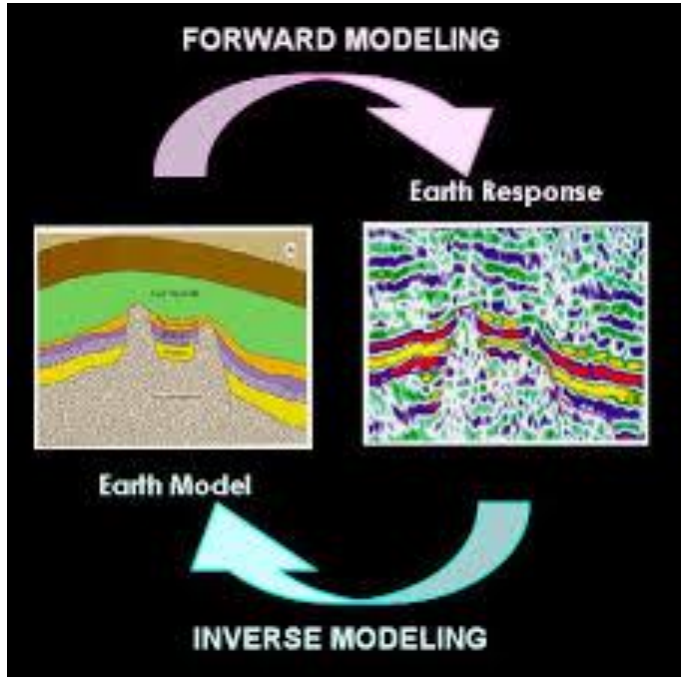
Inverse problems

Given observations / measurements, find PDE parameters, initial conditions, or the forcing functions that produced them

- usually ill-posed, requiring e.g. regularization
- often tackled via multiple forward solves

Applications:

- MRI reconstruction
- earthquake location
- optimize material properties
- Find optimal control inputs



Thanks Everyone!